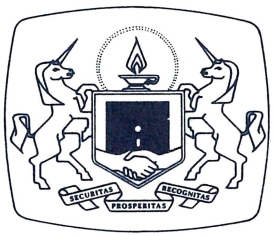
The Bug™

COPYRIGHT © 1981 M.H. RIVNER
LICENSOR: SENSIBLE SOFTWARE

A Product of



The Software Guild

SYSTEM REQUIREMENTS

Apple® II, II+ or Franklin Ace™ 1000
24K RAM • DOS 3.3
One Disk Drive • Game Paddle (optional)

PACKAGE CONTENTS

The Bug Diskette
Instruction Manual
Ready Reference Card
Warranty Card

OVERVIEW

Swat! Kill pesky Assembly Language bugs dead! Works better than a rolled up printout! The Bug is specially formulated to be safe and effective for use on any microcomputer program. It is a control program that displays the source code of each instruction as it is executed, as well as the contents of registers and memory. Execution can be commenced from any point, run fast through portions known to be bug free, and slowed to step-by-step at trouble spots. The run can be automatically halted at points of interest; commands, registers, and data can be changed, and the run continued. The modified program can be stored to disk. At last, a miracle product that lets you walk through a program until you step on The Bug!

This product is being sold under a license from Sensible Software.

The Bug Manual v.1
February 1983
Edited by The Software Guild, Inc.

Apple is a registered trademark of Apple Computers, Inc.
Franklin Ace is a trademark of Franklin Computer Corp.
Softsmith is a trademark of Softsmith Corp.
The Software Guild is a trademark of The Software Guild, Inc.

Table of Contents

Page

System Requirements ..... Front cover
Package Contents ..... Front cover
Overview ..... Front cover

Section One—Introduction 5
Warranty Card 6
Important Tips 7
Getting Started 7

Section Two—A Description Of Commands 9
Execution Commands 9
Set Program Counter and Go: G 9
Go Fast: F 10
Go One Step At A Time: S 11
Interrupt Execution: Escape 11
Controlling The Speed Of Execution 11
Conditional Fast Execution: J 11
Controlled Display: Q 12
Instructional Display: I 12
No Instructional Display: N 12
Set a Memory Breakpoint: B 12
Set The Register Testing Breakpoint: T 12
Display Commands 13
Breakpoint Display: W 13
Display Registers: D 14
Display A Small Part Of Memory: P 14
Extended Memory Dump: Z 14
List Disassembly: L 15
Print Hardcopy: C 15
Text Mode: Control T 15
Graphic Mode: Control G 16
Display Primary Page: Control F 16
Display Secondary Page: Control S 16
Normalize Display: Control N 16
Display $3C-$3D or $3E-$45: & or &, P 16

Modification Commands 16
Reset Stack Pointer: ! 16
Modify Register: R 17
Modify Memory: @ 17
Write Enable For Language Card: Control W 17
Protect Language Card: Control P 17
Checksum Against The Bug: Control K 17
Commands That Exit Or Restart The Bug 18
Exit To Basic: E 18
Access Monitor: M 18
Enter A DOS Instruction: Control D 18
Reset 18
Warmstart: & or CALL 1013 or Control Y 18
Quit The Bug: Control Q 18

Section Three—A Demonstration Of The Commands In Use ... 19

Section Four—Precautions 35
Using The Mini-assembler 35
Entering Multiple Commands 35
Special Considerations for Running The Bug 36
Memory Locations Used By The Bug When It Boots 38
Loading The Bug Onto The Language Card 38

Appendix I: Command Format Summary 39
Appendix II: Address Locations Of The Bug 42

Section One
Introduction

The Bug will make assembly language programming more enjoyable and efficient. It simplifies and hastens the process of debugging. Experienced programmers will appreciate spending more time on the creative, conceptual aspects of programming rather than the tedious work of debugging. Novice programmers will find The Bug is an excellent learning aid since a command can be displayed with its actual result, thereby creating a tutorial from the debugging process.

The Bug is a method of analysis designed to accommodate the needs of assembly language programmers. The program to be studied, which will be referred to as "your program", is loaded into computer memory alongside The Bug. The commands of your program are executed correctly, but at different rates. You instruct The Bug to slow, hasten, or stop the rate of execution. When your program is controlled by The Bug, all values of pointers, addresses, and data are the same as those values under normal running.

Controlling the rate of execution is useful because The Bug can display on screen the command that is being executed, as well as the values of registers and pointers. For example, you might execute one step at a time. By following the commands as they are displayed, you trace the path of your program logic as it is actually run, rather than attempting to extrapolate from print-out sheets. Therefore, The Bug provides routine information without guesswork, tedium, or error.

In addition to illustrating what your program does, The Bug can also demonstrate what would happen if certain values or commands were changed. It allows you to stop the running of your program, make modifications, then continue running your program with the new values in place. Your modified program can then be stored.

You instruct The Bug from your keyboard by entering commands that are typically named by a single character. Many commands can accept further parameters, such as an address. These parameters are part of the command entry and are typed on the same line. Your "Ready Reference Card" and the identical Appendix I list all commands, their format and a very short description of each. You may also choose to use a game paddle to control execution rates.

There are five categories of commands:

- Execution commands tell The Bug to begin running your program at a specified rate.
- Speed and Breakpoint commands choose the execution rate and stop the program run when it reaches a situation or position which is of interest to you.
- Display commands choose what information you are shown on the display screen.
- Modification commands allow you to change the contents of registers or memory.
- Restart or exit commands make BASIC and DOS available.

WARRANTY CARD

Please take a moment to complete and mail the enclosed Warranty Card. By having your name on file we will be able to provide better customer service, including information on program updates and replacement of damaged diskettes.

When you purchased this Softsmith computer program, you chose the trademark of software quality. The distinctive package assures you that the program inside is useful, functions smoothly, and is user-friendly. Consult your local computer dealer for a complete catalog of programs and a demonstration of any of our software products.

IMPORTANT TIPS

Follow these tips to minimize problems in running this and many other programs on your microcomputer.

Handle your diskettes carefully and hold them with your fingers on the labeled edge. Always keep them in their special sleeves while not in use.

Do make sure the shift lock light is lit if running this program on the Franklin Ace. This program recognizes only capital letters.

Do not open the disk drive door while the "in-use" light is on. This may damage your diskette and render it unusable.

Do not press [CTRL-C] **while the "in-use" light is on.** Results are unpredictable and damage to your program may result.

Do not press [BREAK] **while the program is running.**

Do not try to make a copy of this disk. Refer to How To Obtain A Backup Copy, above.

GETTING STARTED

To get started with The Bug follow the usual steps of booting.

1. Turn on your monitor and turn off your computer.

2. Put The Bug program diskette in drive #1, with the label up.

3. Turn the power switch of your computer on. You will hear a "beep" from the computer and the "in-use" light on the disk drive will come on. This means that the program is loading.

You will now see the Softsmith logo page and The Bug title page appear on the screen. If you don't, please try turning your computer off briefly, and then on again.

Booting does not automatically enter The Bug main program into memory, but only a small loader program. The reason is that eventually both "your program" and The Bug will simultaneously be in the computer memory. You must be careful to prevent them from interfering with one another. Therefore, the booting operation lets you specify a page location in memory where The Bug can be safely loaded.

After booting you are prompted by the message ENTER PAGE BOUNDARY IN HEX? If you just press [RETURN] The Bug will be loaded below HIMEM which is then reset. Otherwise specify a page by entering the two most significant digits of the hexadecimal address. It must be at or above $1000, and you must reserve $DFF bytes above it because The Bug is about 4K long. The validity of the address is not checked, so it is up to you to protect the DOS. There is a third alternative; you may enter [L] as the page boundary. This will load The Bug onto the language card from $D000 to $DDFF. Please refer to Section Three, Loading The Bug Onto The Language.

Since you are just learning, just enter the default [RETURN], as the page boundary. Next you will be asked whether you want to LOAD DEMO PROGRAM? Once you have learned how to use The Bug you will always answer [N] or [RETURN], but for now answer [Y]. The demonstration programs which are thereby loaded will be referred to in Section Four of this manual. Incidentally, if your computer keyboard has a shift lock, now is the time to make sure it is on.

Next The Bug checks whether a game paddle is connected to the system. If one is connected, you will see the prompt GAME PADDLE? asking how you intend to use the paddle. If your program uses the paddle, then perhaps The Bug should not. You would respond NO. Otherwise, respond YES or [RETURN] to activate the paddle. In that case the paddle works with commands from The Bug to control the rate at which your program is executed.

Next the screen will be divided into two sections by a dotted line. The upper part of the screen is reserved for the output of your program. The lower portion of the screen is where The Bug will be displaying its "disassembly" information. The word "disassembly" will be used in this manual to refer to the presentation of source codes and register values.

In the lower portion of the screen you will see an inverse asterisk. This will always be your prompt, asking you to enter any of the commands that The Bug makes available to you.

Section Two of this manual tells you what these commands are and how to use them. Section Three gives you some examples of commands in use. Section Four mentions some peculiarities of The Bug that may influence how you use it. The Appendices provide reference information you will need.

Section Two
A Description Of Commands

Commands by which you instruct The Bug can be divided among five categories of function, namely: execution, display, speed control, modification, and exit. You will quickly become fluent with the approximately thirty-five available commands, since most are named by a one-letter mnemonic.

Along with the command name, you will sometimes go on to provide a numerical parameter, or operand. All numbers will be interpreted as hexadecimals, with which you must be familiar. The "format" of a command decides whether such parameters are required, optional, or not allowed. Your Ready Reference Card and the identical Appendix I illustrate these formats. The command formats are clearly described throughout this manual.

The command interpreter for The Bug ignores blank spaces. You may choose to use blanks for clarity. This manual provides examples in both ways.

You must press [RETURN] to complete the entry of commands. More than one command can be entered on a line, and a single [RETURN] will enter all of them. Please refer to Section Four, Entering Multiple Commands. After gaining more experience with The Bug you will find this feature helpful. For example, some commands reset their address pointer, so entering the same command repeatedly will cause consecutive executions.

This manual assumes that you know about control characters. For example, when instructed to press [CTRL-F], this implies you should depress [CTRL] and while holding it down, press [F].

EXECUTION COMMANDS

The following commands will initiate execution of your program by The Bug. The commands differ in the resultant display and execution rates.

Set Program Counter and Go: G

This command is the primary way to go on with the debugging process. Press [G], type in the address (optional), then type [:] and a step count (optional), then press [RETURN]. The Bug will

begin executing your program. The addressing option is used to set the program counter and therefore the starting point of execution. The colon must be present only if the step count option is used. The step count is a number $FF or smaller, and it causes execution to be halted after the specified number of steps.

G is the debugging command you will use the most. It is the only way to set the program counter, and it runs your program. Execution initiated by G obeys all other debugging commands. The run will continue until one of the following conditions is satisfied: the step count is zero, a BRK instruction is encountered, an RTS instruction results in a stack overflow, an RTS with no corresponding JSR is encountered, a breakpoint is satisfied, or the [ESC] button (or the paddle button) is pressed.

When execution is halted, the values of the Register Display Line are shown on screen. This display includes the contents of all 6502 registers, the address accessed by the last instruction executed, the contents of that address, and the ASCII representation of the contents.

Go Fast: F

Use this command to initiate execution that is faster than the standard go command. Press [F], type the completion address, type [:] and a loop count (optional), then press [RETURN]. The program will run at fast speed until it reaches the completion address. The optional loop count specifies the number of times the instruction at the completion address is executed before the program run is halted. The colon is required only if the loop count is used.

If a certain portion of your program is known to be bug free, you would want to run that part fast. Execution initiated by F is not disassembled, other commands from The Bug are not considered, and execution occurs at approximately the full speed your program would normally run. The [ESC] key will not terminate the program run. Breakpoints will not be checked for, with the exception of the conditional breakpoint, which is checked only if and when the loop counter is incremented, at the completion address.

In order for this command to operate properly, the completion address must be in RAM. Also, the instruction there must be executed by your program, so that control can return to The Bug. If control is lost, use the [RESET] option described in this section, the Exit Commands chapter.

In order for the F command to work, three memory locations starting with the completion address are modified. If your program modifies or references these locations during fast execution, unpredictable results will occur. When the fast execution is completed, the original contents of these locations are restored and the Register Display Line is shown.

Go One Step At A Time: S

Use this command to execute a single step at a time. Press [S], type [:] and a step count (optional), then press [RETURN]. A single step or the number of steps specified in the optional step count will be executed. As an instruction is executed, the disassembly and the Register Display Line are shown on screen. This command obeys all The Bug commands in the same manner as G.

Interrupt Execution: Escape

Use this command to escape an execution run. Press [ESC]. The run will be halted, provided it was initiated by G or S, and not by F.

CONTROLLING THE SPEED OF EXECUTION

You may control the speed of execution of your program by using the following commands, then initiating execution by the command G.

Certain critical stretches of the execution of your program will be of interest for an analysis, while other portions will not. You will want to execute the commands of these critical stretches a single step at a time, but execute the less interesting portions automatically and at a normal rate. For example, you can flag the interesting stretches by using a breakpoint command. While executing your program, The Bug checks whether the breakpoint has been reached, and if so, normal execution is stopped.

Conditional Fast Execution: J

This command checks the stack pointer with each JSR instruction and executes at fast speed, like F, while the pointer is under a specified value. Press [J], type [:] and the value to be tested for. Execution speed returns to normal G rate when a RTS instruction restores the stack pointer to the entered value.

Since J modifies memory like the F command, the same restrictions apply. Also, subroutines that modify their return address will not operate with this command.

The J command is a toggle. To disable it, enter the command with no operands.

Controlled Display: Q

Use this command when you want execution to occur and be displayed on screen at a readable speed. Press [Q], then press [RETURN]. All further execution initiated by a G command will be at a controlled rate. If the game paddle was activated for use by The Bug, then it can be used to control the speed. Just turn the potentiometer. If the game paddle is not being used, then a preset controlled display speed is used.

Instructional Display: I

Use this command to get the standard instructional display. Press [I], then press [RETURN]. When execution is initiated with the G command, all instructions will be shown disassembled before the instruction is performed. The display will be too fast to read.

No Instructional Display: N

Use this command when no display is desired during execution. Press [N], then [RETURN]. Execution is initiated by the G command, and will run faster than with the instructional display. When execution is halted, the Register Display Line will appear on screen.

Set A Memory Breakpoint: B

This command allows you to establish a breakpoint at a specific address in ROM or RAM. Press [B], type an address (optional), press [:] and a breakpoint number (optional), then press [RETURN]. Use the address option to reset the default address value. Use the numbering option to identify a breakpoint.

Up to eight breakpoints can be established. They are numbered automatically, starting with the lowest free number, going one to eight, then wrapping around. You may optionally specify the identifying number, which is how you would overwrite or erase a breakpoint. To erase all breakpoints, press [B], then [RETURN] with no operands.

Set The Register Testing Breakpoint: T

Use this command to establish a test of register values. Press [T], press the letter of the register, type the symbol(s) of the condition, type the value to be tested for, then press [RETURN]. The letter of the register is: [A] for Accumulator, [X] for X register, [Y] for Y register, or [S] for the Stack Pointer. The symbols of con-

dition are [=], [<], and [>], for equals, less- than, and greater-than. Symbols may be combined for less-than- or-equal and greater-than-or-equal.

The position of status register flags is also a part of the register testing breakpoint. Press [T], [P] command, then type the flags with values, then press [RETURN]. Separate one flag with its value from the next by a comma. Flags are designated [Z] for the zero flag, [V] for overflow, [C] for carry, [D] for decimal, [I] for interrupt, [B] for soft break, and [N] for sign flag. Flag values are [0] or [1]. All the status flags you want to test for must be included in one TP command, and flags not mentioned are cancelled from the test.

All test conditions must be satisfied for the breakpoint to occur. Separate commands are used to establish each register's test, but they are all part of the single register testing breakpoint. The T command with a designation will overwrite the conditions for that register, but the previously established tests for other registers are not changed. To cancel a register's test, press [T], then the letter of the register, then [RETURN]. To cancel all old tests, enter the T command with no designations; that is, press [T], then press [RETURN].

Program execution initiated by the G command will halt when all the register conditions set with the T command are satisfied. The conditions set can be examined with the W command. Execution initiated by the F command with its optional loop count set will also be halted if T conditions are satisfied at the completion address. Execution initiated by F alone is not affected by breakpoints.

DISPLAY COMMANDS

When The Bug is running, you will usually want to see the disassembled source code of your program as it is being executed, the contents of registers, and some of your program's output. These constitute the standard screen display. Sometimes you may want to change the nature of the display by using the following commands. They either call for special information or change how the display is being used.

Breakpoint Display: W

Use this command to check where or when the breakpoints have been set. Press [W], then [RETURN]. The screen will display a list of all the memory location breakpoints and the register conditions breakpoint.

Display Registers: D

Use this command to display current information relevant to debugging. Press [D], then [RETURN]. The following information will be displayed: the contents of the 6502 registers (A, X, Y, P, and S), the program counter (PC), the value of every bit in the status register (C, Z, I, D, B, V, S), the address accessed by the last instruction (when applicable), and the contents of that address. This information is called the "Register Display Line."

Display A Small Part Of Memory: P

Use this command to get a partial memory dump. Press [P], then type an address (optional), then press [RETURN]. The values of up to eight memory locations will be displayed on screen. Specifically, the memory locations shown are those from the current address pointer of P to the next address divisible by eight. The current address pointer can be set by using the address option, and it is incremented by one as each memory location is printed on screen.

Extended Memory Dump: Z

Use this command to print a zone of the memory consisting of 120 contiguous memory locations. Press [Z], then type an address (optional), then press [RETURN]. The optional address specifies where the memory dump is to start.

If no address is specified, then the dump begins at a default address, usually the one set by the last Z command. The value of the default address is updated, so when the command is entered repetitively, the next zone of memory will be dumped.

Here is a sample of an extended memory dump:

```
0B10-  D4 C5 D8 D4 A0 D7 C9 D4   TEXT WIT
0B18-  C8 A0 CD D3 C2 A0 D3 C5   H MSB SE
0B20-  D4 0A C9 D3 A0 C4 C9 D3   T IS DIS
0B28-  D0 CC C1 D9 C5 C4 A0 C9   PLAYED I
0B30-  CE A0 CE CF D2 CD C1 CC   N NORMAL
0B38-  A0 D6 C9 C4 C5 CF AE A0   VIDEO.
0B40-  54 45 58 54 20 57 49 54   TEXT WIT
0B48-  48 20 4D 53 42 20 43 4C   H MSB CL
0B50-  45 41 52 45 44 20 49 53   EARED IS
0B58-  20 44 49 53 50 4C 41 59   DISPLAY
0B60-  45 44 20 49 4E 20 49 4E   ED IN IN
0B68-  56 45 52 53 45 20 56 49   VERSE VI
0B70-  44 45 4F 20 42 55 54 20   DEO BUT
0B78-  53 50 41 43 45 53 20 41   SPACES A
0B80-  52 45 20 4E 4F 54 2E 20   RE NOT.
```

The left column lists every eighth address in the zone. The middle columns are the hexadecimal values in those memory locations. The right column gives the ASCII conversion of those values. True ASCII characters, with MSB cleared, will be displayed in inverse video.

List Disassembly: L

Use this command to list ten instructions in their original assembly language source code. Press [L], then type an address (optional), then press [RETURN]. Ten instructions will be listed in the order of consecutive addresses. There is no reason to assume they would be executed in that order. The address of each instruction will appear to the far left, then the machine language for the command and operand, then the source code and operand on the right. A sample list appears below:

```
0B00- A0 20    LDY  #$20
0B02- A2 14    LDX  #$14
0B04- 88       DEY
0B05- D0 FD    BNE  $0B04
0B07- CA       DEX
0B08- D0 FA    BNE  $0B04
0B0A- 60       RTS
0B0B- 00       BRK
0B0C- 00       BRK
0B0D- 00       BRK
```

Large portions of code may be disassembled by entering more than one L command on a line. For each L entered, ten instructions will be disassembled.

Print Hardcopy: C

Use this command to get a paper copy of The Bug's output. Press [C], then [RETURN]. This will toggle (activate or deactivate) the printer connected to slot #1. If slot #1 is empty, this command will have no effect. Printers connected to other slots are not affected. Since The Bug has an output routine separate from that of the user program, this command will have no effect on user program output.

Text Mode: Control T

This command puts the screen into text mode. Enter [CTRL- T], and complete the entry with [RETURN]. This is equivalent to "LDA $C051".

Graphic Mode: Control G

This command puts the screen into graphic mode. Enter [CTRL-G], then [RETURN]. This command allows you to see the video display that your program has plotted. It is equivalent to "LDA $C050". After an instruction is executed, the display screen is reset to the text mode.

Display Primary Page: Control F

This command displays the first, or primary, page. Enter [CTRL-F], then press [RETURN]. It is equivalent to "LDA $C054".

Display Secondary Page: Control S

This command displays the secondary page. Enter [CTRL-S], then [RETURN]. This is equivalent to "LDA $C055". The display will be automatically reset to primary page after an instruction is executed.

Normalize Display: Control N

Use this command to set the display back to its normal mode. Enter [CTRL-N], then press [RETURN]. This sets the display to text mode and primary page. This is equivalent to "LDA $C051, LDA $C054".

Display $3C-$3D or $3E-$45: & or &, P

Enter [&], then press [RETURN] to display the values swapped to memory locations $3C-$3D.

Enter [&], press [P], then [RETURN] to display the values swapped to locations $3E-$45.

MODIFICATION COMMANDS

When using The Bug, you may wish to skip through a loop by bumping a register value, or you might want to check what route your program would take if certain changes were made. You can make such changes by using the modification commands below. These commands allow the user to change the contents of any register or memory locations, including language card memory.

Reset Stack Pointer: !

Use this command to reset the stack pointer back to $FF. Press [!], then press [RETURN].

Modify Register: R

This command allows the user to modify a 6502 register. Press [R], press the letter designating which register is to be modified, press [:], type the hexadecimal value to be placed in the register, then press [RETURN]. Use the following letter designations: A for Accumulator, X for the X register, Y for the Y register, S for the Stack Pointer, and P for the Status register.

Modify Memory: @

This command modifies the contents of memory at a specified address. Up to eight changes can be made. Press [@], type an address (optional), press [:], type the list of values, then press [RETURN]. Use the addressing option to reset the current default address of @ which is then incremented as memory is modified. The colon must be present, even if no address precedes it. The values in the list must be hexadecimal numbers, and they must be separated from each other by a space or comma. When two byte word values are entered, The Bug automatically inverts the high order and low order bytes, so the value will be placed in memory correctly.

Write Enable For Language Card: Control W

With this command you can write to the language card. Enter [CTRL-W], then press [RETURN]. This command need only be issued once; then addresses on the language card, like addresses of normal memory, may be modified with the @ command.

The DOS pointers that point to the current version of BASIC are not modified. You may switch to a different version of BASIC if the language card is turned on. Consult your language card documentation.

Protect Language Card: Control P

Use this command to protect your language card from being written on. This command is equivalent to "LDA $C080". It prevents further modification of language card memory. Consult your language card documentation.

Checksum Against The Bug: Control K

Use this command to check that you have not accidentally modified The Bug program image in memory. Enter [CTRL-K], then press [RETURN]. Two forms of checksum will be performed on the main program, and an error message will be displayed if appropriate. The Bug should be reloaded if an error is detected.

Exit To BASIC: E

Use this command to exit to BASIC. Press [E], then [RETURN]. While in BASIC, you may execute any BASIC or DOS command. Return by using a warmstart command. If either the FP or INT command is entered, HIMEM will be reset and The Bug will not be protected.

Access Monitor: M

Use this command to transfer to the monitor. Press [M] and [RETURN]. This puts you in monitor. [CTRL-Y] returns you to The Bug's warmstart routine; then enter 3F5G or 3F8G.

Enter A DOS Instruction: Control D

Use this command to enter a DOS instruction without exiting The Bug. Enter [CTRL-D], type the DOS command, then press [RETURN]. The instruction will be executed.

Reset

To return to the coldstart routine of The Bug, press [RESET]. With an autostart ROM you will be returned directly and the stack pointer will be reset to $FF. In addition a number of internal pointers of The Bug will be reset. You must not have modified the values contained at $3F2-$3F4, since these point to the coldstart routine.

If the monitor ROM is used, press [RESET] to access the monitor. Use the monitor G command to execute the instruction starting at the sixth memory location in The Bug. In a 48K Apple with The Bug at HIMEM, this location is $8905. The command 8905G will get you back.

Warmstart: & or CALL 1013 or Control Y

Use these commands to do an unconditional jump to The Bug warmstart routine. Enter [&] from Applesoft. Enter CALL 1013 from BASIC. The contents of $3F5-$3F7 must not have been changed. Enter [CTRL-Y] from the monitor. This instruction will not work correctly if the contents of $3F8-$3FA are modified.

Quit The Bug: Control Q

This command quits a session with The Bug. Enter [CTRL-Q], then press [RETURN]. This removes the program from memory. The page number vectors are restored to their normal values, and a DOS coldstart is performed.

Section Three
A Demonstration Of The Commands

This section gives examples of The Bug commands in use. There are two demonstration programs on your disk which are loaded into memory at $B00-$CB4, provided you answer [Y], "yes", to the booting prompt LOAD DEMO PROGRAM? These will be used to illustrate how The Bug is used. The listings in this manual will sometimes show comments in the right hand column, but these would not actually appear on screen.

The first program is a simple timing loop. Use listing command, starting at address $B00, to see the program instructions.

```
[*] L B00

0B00- A0 20   LDY  #$20     • X and Y reg reset
0B02- A2 14   LDX  #$14
0B04- 88      DEY           • Counting loops
0B05- D0 FD   BNE  $0B04
0B07- CA      DEX
0B08- D0 FA   BNE  $0B04
0B0A- 60      RTS
0B0B- 00      BRK           • Not part of program
0B0C- 00      BRK
0B0D- 00
```

Execution of this program may be initiated with the following command:

```
[*] G B00
```

Each instruction is displayed as it is executed, but the display scrolls too rapidly for you to read. When the RTS instruction at $BOA is executed, The Bug's monitor prompt will return. You may also halt execution by pressing [ESC] or the paddle button. It takes a long time for this short program to run because The Bug executes many lines of its internal code for each instruction of your program.

When execution is halted, the Register Display Line appears on screen, which tells you the values of the program counter, the 6502 registers, the status flags, the address accessed by the last instruction executed, and the final contents of that address.

You can slow down the display so it can be read.

[*] Q

[*] G B00

Each instruction is again displayed, but at a readable rate. If the game paddle is active, then the speed of display may be varied by turning the potentiometer. This allows you to examine closely the parts you are interested in but pass quickly through the rest. A preset rate of display is used if the game paddle option is not active.

Program execution may be terminated with either the [ESC] key or the paddle button. The Register Display Line will appear on screen.

Instruction execution can be speeded up by the N command. When this command is active the instructions will be executed without being disassembled and displayed.

[*] N

[*] G B00

The Register Display Line will be shown when execution is halted.

An optional instruction counter may be used with the G command.

[*] I

[*] G B00:5

This set of instructions first resets The Bug to the standard display mode and then executes five instructions starting at $B00. The instruction counter may be any number from $1 to $FF. You may still halt execution with [ESC].

When no address is entered with the G command, the default address used is the next executable instruction. You may use the default address and yet specify an instruction count.

[*] G:1

The above executes the single next instruction. This gives the same result as:

[*] S

You may also use the S command to execute more than one instruction.

[*] S:A

In this example 10 instructions are executed and displayed. The [ESC] key and breakpoints can also terminate execution. A starting address cannot be specified and must be set using G. The N and Q commands have the same effects under the S command as they do with G.

[*] N

[*] S:10

[*] I

In this example 16 instructions are executed. Since the N command is active, only the Register Display Line appears on screen, but it does so after each instruction. Finally the I command reactivates the standard display mode.

If a printer is attached to slot #1, you can get a paper copy of your operations.

[*] C

[*] G:12

The first command directs all output to the printer. The second command executes $12 instructions. Depending on the printer interface used, output may be displayed on the screen as well. All output is directed to the printer until another C command is issued to toggle the copy option off.

[*] C

You may modify the values in your registers. For example, restart the demo program and execute three instructions.

[*] G B00:3

The program counter is now equal to $B05, the Y register equals $1F, and the X register equals $14. Use the following commands to modify the contents of these registers so X equals 2 and Y equals 3.

```
* RX:2

* RY:3

* D
```

The D command puts the Register Display Line on screen, and the values there confirm that the change has been made.

Execution is now restarted.

```
* G
```

The program runs through quickly because the outer loop is executed only twice.

The Bug may be used to modify a program. You will use one of two methods, depending on whether or not you have a mini-assembler.

The monitor is entered with the M command.

```
* M
```

Set the bank select. You should read Using The Mini-assembler, Section Four, if you have trouble. The bank select is located at C084, so reference it while in the monitor.

```
* C084
```

Next the mini-assembler is entered.

```
* F666G
```

Now you make the modifications to your program.

```
! B00:LDX #$14
0B00-    A2      14     LDX    #$14
! LDY #$20
0B02-    A0      20     LDY    #$20
! B08:BNE $B02
0B08-    D0      F8     BNE    $0B02
! $FF69G
*^Y
```

The $FF69G instruction returns you to monitor, and the [CTRL-Y] instruction returns you to The Bug. The modified program can now be listed.

```
[*] LB00

0B00- A2 14    LDX  #$14
0B02- A0 20    LDY  #$20
0B04- 88       DEY
0B05- D0 FD    BNE  $0B04
0B07- CA       DEX
0B08- D0 F8    BNE  $0B02
0B0A- 60       RTS
0B0B- 00       BRK
0B0C- 00       BRK
0B0D- 00       BRK
```

If you do not have the mini-assembler, you may still change the program by manually assembling the codes. The following command lines will make the same modifications as the above method.

```
[*] @B00:A2,14,A0,20

[*] @B09:F8

[*] LB00
```

The new program may now be stored on disk with a BSAVE command, and/or executed directly from memory. To save, use the following sequence of commands, but first remove The Bug program disk and insert a standard DOS disk. As when entering any DOS command, you use the E command to exit.

```
[*] E

]BSAVE BUG.DEMO.I,A$B00,L$B

] CALL 1013
```

You may run the modified program.

```
[*] GB00
```

The new version takes considerably less time to execute than the original, since the Y register is not reset to $FF each time the instruction at $B08 occurs.

The P command displays a partial memory dump of the hexadecimal values of up to 8 memory locations.

```
[*] PB15

0B15- D7 C9 D4
[*] P

0B18- C8 A0 CD D3 C2 A0 D3 C5
```

The Z command displays a large zone of memory and its ASCII translation. The sample printout shown in Section Two, Extended Memory Dump is a zone of the demo program, and you can display it on screen.

```
[*] ZB10
```

Note that the text from $B10 to $B3F is in normal video, as these values all have their MSB set. The text from $B40 to $B87 is in inverse video since these values have their MSB cleared.

To stop program execution at location $B07, set a breakpoint.

```
[*] B

[*] B B07
```

The first B command clears all register breakpoints, and the second one sets a breakpoint at $B07. Now start execution at the beginning of the program.

```
[*] GB00
```

The first loop will run, and when the instruction at $B07 is encountered, execution will stop.

```
[*] G
```

Execution is restarted and will continue until the instruction at $B07 is again reached.

The breakpoints that are set may be displayed.

```
[*] W
```

Next the T command will be used to set up a register breakpoint that will cause program execution to halt when the Y register equals 6.

```
* TY=6

* B

* G
```

The first command sets the register testing breakpoint. Previously set memory breakpoints are cleared by the B command. Execution will stop at $B05. Note that the Y register is equal to 6.

```
* G
```

Only a single instruction gets executed since it does not change the Y register, which still satisfies the breakpoint conditions.

```
* G
```

This time, many instructions are executed since the next instruction modifies the Y register. Execution will continue until the Y register again equals 6.

Now set up The Bug so that execution halts only when both the Y register equals 6 and the X register equals 5.

```
* RX:10

* TX=5

* G
```

The first instruction resets the X register to $10. Next a breakpoint condition is set for the X register. A breakpoint condition is already set for the Y register. Execution is then initiated and continues until $B05, when the conditions set for the X and Y registers are both satisfied. All conditions established must be satisfied for execution to halt. As many register conditions may be set as desired. Registers not set are not tested for. A display of the current breakpoint conditions can be obtained.

```
* W
```

The breakpoints are displayed on screen. To remove the X register breakpoint condition, it would be set with no operands. Then execution would again depend on only the Y register.

To cancel the register condition testing breakpoint, clear all the breakpoint conditions.

```
* T
```

Now set a breakpoint that will be satisfied every time the Z flag is set to 1.

```
* TP,Z1

* G
```

Execution will continue until $B05, at which point the Y register was decremented to zero by the previous instruction, setting the Z flag. Now set the breakpoint so that execution is stopped when the sign flag is cleared.

```
* TP,S0

* W
```

The sign flag is the only active condition for the breakpoint, since setting it cleared the Z flag breakpoint. You may set both with the following command.

```
* TP,S0,Z1
```

Now clear all register breakpoint conditions.

```
* T
```

You can execute the program at full speed.

```
* GB00:1

* FB0A
```

The program is executed in a very short time. Breakpoints are not checked. The initial G command was used to establish the address of the next executable instruction as the beginning point of the program run. The F command cannot set this address. The program runs until the specified completion address is reached. That address becomes the next executable instruction. In this case it would be $B0A.

An optional loop count can be specified.

```
* GB00:1

* FB08:2
```

In this case execution starts at $B00 and continues until the instruction at $B08 is executed two times.

The register condition breakpoint will be checked for when the loop count is used, but this check will occur only at the time the instruction at the specified completion address is executed.

```
* TX=8

* FB08:F
```

In this case the instruction at $B08 is not executed 15 times since the X register is equal to 8 at the 10th time the instruction is executed. The X register is only checked when the instruction at $B08 is executed. Note that the starting address in this instruction has been set by the previous instruction to be $B08.

To specify an address that will never be executed is an error.

Do not do this.

```
* T

* GB00:1

* FB0B
```

Location $B0B is never executed, and as a result control will not return to The Bug. When the RTS occurs, stack overflow will occur, taking you to the monitor. You can return by pressing the RESET key. With a monitor ROM you must execute the 6th location in The Bug. The address of this location can be found at $3F2-$3F3. In a 48K Apple with The Bug located at HIMEM this location is $8905. The monitor command 8905G will return you to The Bug.

The second demonstration program is located from $B90 to $CB3. It is somewhat more complex and will be used to demonstrate the command J. This program asks a few questions to which you respond, then the bell will ring 16 times, and finally the speaker will click 256 times. The interval between clicks will depend on the setting of game paddle zero. The following command lists the program.

```
* LB90,L,L,L,L,L,L,L
```

```
0B90  -A0  05      LDY   #$05      5 spaces
0B92  -20  8E  FD  JSR   $FD8E     CROUT
0B95  -88          DEY
0B96  -D0  FA      BNE   $0B92
0B98  -B9  3B  0C  CLDA  $0C3B,Y   Loc of quest #1
```

```
0B9B  -08           PHP          Save status
0B9C  -09  80       ORA  #$80    Set MSB
0B9E  -20  F0  FD   JSR  $FDF0   COUT1
0BA1  -C8           INY          Next character
0BA2  -28           PLP          Stop if neg
0BA3  -10  F3       BPL  $0B98   Loop
0BA5  -20  6F  FD   JSR  $FD6F   GETLIN
0BA8  -20  8E  FD   JSR  $FD8E   CROUT
0BAB  -A0  00       LDY  #$00    Res char pntr
0BAD  -B9  54  0C   LDA  $0C54,Y Loc of nxt output
0BB0  -08           PHP          Save sta
0BB1  -09  80       ORA  #$80
0BB3  -20  F0  FD   JSR  $FDF0   Output char
0BB6  -C8           INY          Next
0BB7  -28           PLP          Stop if neg
0BB8  -10  F3       BPL  $0BAD   Print loop
0BBA  -A0  00       LDY  #$00
0BBC  -B9  00  20   LDA  $200,Y  Input buf
0BBF  -C9  8D       CMP  #$8D    End at CR
0BC1  -F0  06       BEQ  $0BC9
0BC3  -20  F0  FD   JSR  $FDF0   Output char COUT1
0BC6  -C8           INY          Next char
0BC7  -10  F3       BPL  $00BC   Print loop
0BC9  -A0  00       LDY  #$00
0BCB  -B9  57  0C   LDA  $0C57,Y Next string
0BCE  -08           PHP          Save sta
0BCF  -09  80       ORA  #$80    Set MSB
0BD1  -20  F0  FD   JSR  $FDF0   Output char
0BD4  -C8           INY          Next char
0BD5  -28           PLP
0BD6  -10  F3       BPL  $0BCB   Print loop
0BD8  -20  8E  FD   JSR  $FD8E   CR
0BDB  -A0  00       LDY  #$00
0BDD  -B9  6B  0C   LDA  $0C6B,Y Next string
0BE0  -08           PHP          Save sta
0BE1  -09  80       0RA  #$80    Set MSB
0BE3  -20  F0  FD   JSR  $FDF0   Output char COUT1
0BE6  -C8           INY
0BE7  -28           PLP
0BE8  -10  F3       BPL  $0BDD   Print loop
0BEA  -8A           TXA          X # of char spaces
0BEB  -20  DA  FD   JSR  $FDDA   Output val PRBYTE
0BEE  -A0  00       LDY  #$00
0BF0  -B9  78  0C   LDA  $0C78,Y Next string
0BF3  -08           PHP
0BF4  -09  80       0RA  #$80
0BF6  -20  F0  FD   JSR  $FDF0   Output char COUT1
0BF9  -C8           INY
0BFA  -28           PLP
```

```
0BFB -10 F3    BPL  $0BF0   Print loop
0BFD -20 8E FD JSR  $FD8E   CR
0C00 -A0 10    LDY  #$10    16 times
0C02 -20 3A FF JSR  $FF3A   Bell
0C05 -88       DEY
0C06 -D0 FA    BNE  $0C02   Loop
0C08 -A0 00    LDY  #$00
0C0A -B9 91 0C LDA  $0C91,Y Next string
0C0D -08       PHP
0C0E -09 80    ORA  #$80
0C10 -20 F0 FD JSR  $FDF0   Output char COUT1
0C13 -C8       INY
0C14 -28       PLP
0C15 -10 F3    BPL  $0C0A   Print loop
0C17 -20 8E FD JSR  $FD8E   CR
0C1A -A9 FF    LDA  #$FF    256 clicks
0C1C -8D 3A 0C STA  $0C3A   Save
0C1F -CE 3A 0C DEC  $0C3A   Memory
0C22 -F0 10    BEQ  $0C34   End after 250
0C24 -A2 00    LDX  #$00    Game paddle #0
0C26 -20 1E FB JSR  $FB1E   Read game paddle
0C29 -C8       INY          Paddle val
0C2A -98       TYA          In accm
0C2B -20 A8 FC JSR  $FCA8   Wait loop
0C2E -AD 30 C0 LDA  $C030   Click speaker
0C31 -4C 1F 0C JMP  $0C1F   Loop
0C34 -60       RTS
```

There are more elegant methods for performing these functions, but this program is good for demonstration purposes. The variable and string storage sections of this program can be displayed by the following command.

[*] ZC3A

Now run the program using the G command.

[*] GB90

Note that it runs very slowly. When input is requested, the user must pause between entering each character to give the program a chance to catch up. The bell does not sound like a bell, and the clicks are very far apart. The game paddle input is not read in correctly. Execution can be speeded up with the N command.

[*] N

[*] GB90

Execution is still very slow. Many of the functions used in this program must execute at full speed to work correctly. The J command is designed for this purpose. You may set a value for the stack so that when a JSR instruction pushes the stack below that level, execution will run at full speed. Control will return to The Bug when a RTS instruction restores the stack pointer to the calling level. In the case of this program, set this value to $FF, as all subroutines called by this program are located in the monitor. You know these subroutines work correctly, and there is no need to debug them.

```
* I

* !

* J:FF
```

The I command brings back the standard display mode. The ! command resets the stack pointer to $FF. The J command establishes that conditional full speed execution should be used. You now begin execution.

```
* GB90
```

Execution occurs much faster. String input occurs at normal speed. The bell sounds correctly. The clicks still occur very far apart, but you can hear the effect of turning the game paddle. Only the instructions in the main program are disassembled, and only those disassembled instructions are checked for breakpoints. All subroutines are run at full speed since their instructions are not disassembled and breakpoints are not checked for.

Speed is increased further by using the N command in conjunction with the J command.

```
* N

* G B90
```

The clicking sounds are much faster, and the game paddle has a more noticeable effect.

Now run the program at full speed.

```
* G B90:1

* F C34
```

The clicks are very close together and can be made into a single tone by turning the game paddle.

Now modify the program so that game paddle #1 is used to control the speed of the clicks instead of game paddle #0.

First, list the instructions that need to be changed.

```
[*] LC24

0C24  -A2  00      LDX  #$00
0C26  -20  1E FB   JSR  $FB1E
```

Of the ten instructions actually listed, the first is of interest. It places a 0 in the X register, so that is the paddle number which is assumed by PREAD at $FB1E.

```
[*] @C25:1

[*] LC24

0C24  -A2  01      LDX  #$01
0C26  -20  1E FB   JSR  $FB1E
```

Run the program with Q active, so that instructions will be disassembled and displayed.

```
[*] Q

[*] GB90
```

The speed of program execution is controlled by game paddle #0, if it is active. The speed of the clicks is controlled by game paddle #1.

Do not use the J command when the jump to fast speed occurs in ROM. The following commands would illustrate this.

```
[*] !

[*] I

[*] BB98

[*] GB90

[*] J:FD

[*] G
```

The I command brings back the standard display mode. The J command establishes conditional fast speed execution to occur when a JSR instruction pushes the stack pointer below $FD. Actually, execution will halt at $FDF9 since the JSR at this instruction does push the stack pointer below $FD, but that instruction is in ROM. Fast speed execution cannot occur since locations $FDFC-$FDFE would need to be modified. The error message ADR NOT IN RAM is displayed, and control returns to The Bug. To reset the J command to $FF, you must first reset the stack to $FF. The following commands will accomplish this.

```
* J
* TS=FF
* G
* T
* J:FF
* G
```

The initial J command toggles off the conditional fast speed mode. The T command sets a conditional register breakpoint so execution will halt when the stack pointer equals $FF. Next the G command initiates execution which then is halted when the stack pointer equals $FF. Now the breakpoint is removed with an empty T command. J command is again established, and execution is restarted with G.

The following commands demonstrate how to establish breakpoints. They will divide the program into eight parts, which is the maximum number of memory location breakpoints available.

```
[*] B BAB

[*] B B98

[*] B:1

[*] B BBA

[*] B BC9

[*] B BD8

[*] B BFE

[*] B C00

[*] B C08

[*] W

BRKPT  #1 0BBA
BRKPT  #2 0BAB
BRKPT  #3 0B98
BRKPT  #4 0BC9
BRKPT  #5 0BD8
BRKPT  #6 0BFE
BRKPT  #7 0C00
BRKPT  #8 0C08
```

Remember, breakpoint #1 was previously set. The first 2 breakpoints entered above filled the 2 lowest free locations, namely breakpoints #2 and #3. Then breakpoint #1 was erased with the command B:1; therefore when the next breakpoint entered was placed in the lowest unfilled location, it was breakpoint #1. The other breakpoints filled locations 4 to 8 in sequential order.

```
[*] G B90
```

Execution is started and then halts at $B98 because breakpoint #3 was satisfied.

```
[*] F BA5

[*] G
```

Execution is now stopped at $BAB, the next breakpoint encountered. Now enter a new breakpoint.

```
[*] B C1A

[*] G
```

Execution does not stop at $BBA but rather continues to $BC9. This is because breakpoint #1, which was previously set to $BBA, has been changed to $C1A by the above B command. When there are no free breakpoints, the ninth is the first by wrap around. The next breakpoint entered would overwrite onto breakpoint #2.

These examples should help you become familiar with the program. Your next step is to write a small program and debug it.

Section Four
Precautions

USING THE MINI-ASSEMBLER

To use the mini-assembler you must have INTeger BASIC available in memory. The Bug program diskette has only Floating Point, and not INTeger. Therefore, instead of booting The Bug directly, first boot your system master. When finished, remove the master from the drive and insert The Bug diskette. Type the command BRUN THE BUG [RETURN]. Do not load onto the language card when asked for a page boundary.

Before you can use the mini-assembler you should go to monitor and set the bank select. From The Bug prompt press [M] [RETURN]. The monitor's asterisk prompt will appear. Type C084 [RETURN] to set the bank select soft switch. (Be sure you press zero and not the letter "O".)

The mini-assembler is entered from the monitor. Type F666G [RETURN]. F666 is the mini-assembler location and it is followed by G, the monitor command for "go". The mini-assembler prompt ! will appear.

Your program can now be modified by typing a location, colon, the source code, space, and a value; then finally [RETURN]. For example

```
! B00:LDX #$14
```

You can get back to The Bug by just pressing the [RESET] button. Another way is to type $FF69G [RETURN] to get back to the monitor, then [CTRL-Y] to get back to The Bug.

ENTERING MULTIPLE COMMANDS

You may type multiple commands on a single line. If this is done, be careful to follow the proper command syntax. Each command must be terminated by either a comma or carriage return. In the

case of the @ and TP commands, the value lists can contain commas, so these commands are terminated by a period. When the command mnemonic cannot be mistaken as a hexadecimal digit or a valid part of the previous command, no command delineators are necessary. Any spaces typed in are ignored by the command interpreter, but may be used for clarity.

```
* B A02,B 300,W
```

In the above example each command was followed by a comma.

```
* TY=>3A,TX<FE,TP,C0,Z1.W
```

In this instance the TY and TX are both delineated with a comma, and the TP command is delineated from the final W by a period, since a comma would be incorrect.

```
* LB90LLLLLL
```

In this case a command delineator is not necessary since the letter "L" cannot be mistaken for a hexadecimal digit.

SPECIAL CONSIDERATIONS FOR RUNNING THE BUG

For the 6502 microprocessor, base page locations are efficient, yielding faster execution speed and decreased program size. Base page locations are therefore used by The Bug. All but three of the base page locations used by The Bug are swapped, or saved in another location, when the user's program is executing. Because of swapping, examination of certain locations will not produce the expected values. A list of the locations swapped appears in Appendix II.

Locations $3C to $45 are used by The Bug as address registers, but these locations are swapped, so you need not worry about them. The & command allows examination of the values at these locations during execution of your program.

Three base page locations may not be used by your program. $3A, $3B, and $2F must be reserved for The Bug and are not swapped.

The Bug also has its own set of input and output hooks. The hooks from your program are swapped out and placed in the first five locations of The Bug. If the default relocation value was chosen in a 48K Apple, then the input and output hooks would be placed at $8900 to $8904. These locations correspond to locations $35 to $39. Beware when program execution is stopped and not continued while in DOS. DOS switches the input and output hooks while it

is executing. If you do not manually reset the input and output hooks to their original value, DOS I/O is lost until the hooks stored at $8900 to $8904 are restored. The normal value of these hooks in a 48K Apple starting at $8901 is $BD, $9E, $81 and $9E.

If your program is halted in the DOS I/O routines and you exit The Bug to monitor or BASIC, problems may arise. The problems occur because the DOS input and output routines save the value of the registers and stack pointer. You are free to exit using E or M provided your program is not in a DOS I/O.

The Bug maintains two stacks, one for itself and one for your program. These values are not in their regular location but may be found at their relocation address plus $C00. The Bug will execute instructions that reference locations $100 to $1FF correctly provided indirect addressing is not used. The following example illustrates this.

```
TSX              The val of the SP is in the X reg.
LDA  $100,X      The Bug will modify this instruction so
                 the correct value is used.
LDA  #$10
LDY  #$00
STA  $FF
STY  $FE         $FE-FF contain $100
TXA
TAY
LDA  ($FE),Y     In this case The Bug would not modify
                 the instruction, and the incorrect value
                 would be used.
RTS
```

This should not be a big problem since the stack is rarely addressed indirectly.

Programs that use interrupt-driven routines will be difficult to debug, because when an interrupt occurs The Bug will lose control of execution. This should not be a big problem since interrupt-driven routines are rarely used with an Apple or Franklin computer.

The Bug swaps out the line input buffer, but this swap is transparent. The maximum size allowed for the input buffer is 39 characters. You are not likely to enter a command line longer than that. Using monitor commands corrupts the input buffer.

When either the E or M command is used, the values at locations $36-$39 are replaced with the values found at a displacement of $7C4-$7C7 from the beginning of the program. The locations of other base page values can be found in Appendix II.

MEMORY LOCATIONS USED BY THE BUG WHEN IT BOOTS

Memory locations $200-$DFF are used by the loader program that boots The Bug. Data located at these locations will be destroyed during booting. After booting, you may use this memory without difficulty.

LOADING THE BUG ONTO THE LANGUAGE CARD

During the booting and loading operation you are asked to ENTER PAGE BOUNDARY IN HEX? It is possible to respond "language card" by pressing [L], then [RETURN] as your answer. The Bug will be loaded onto Bank 1 of the language card, between $D000 and $DDFF. This area of the language card is not used by either Applesoft or Integer BASIC. A small program of $FF bytes is placed below HIMEM, which is reset.

When The Bug is located in the language card, you should not use the Control W and Control P commands. A jump to location $D005 will not coldstart The Bug. CALL 1013 will return the user from Integer BASIC.

The small (bank switch) program of $FF bytes may be loaded anywhere in memory above $1000. Respond to the question ENTER PAGE BOUNDARY IN HEX? with a page address for the small program typed on the same line as the [L] for the main program, then press [RETURN]. This will keep HIMEM from being changed.

FOR EXAMPLE: ENTER PAGE BOUNDARY IN HEX? 20L

As a convenience, an option lets you easily load the bank switch program in the upper part of the register buffer. Answer the page question by pressing [L], [X], and then [RETURN]. The main program will be loaded in the language card, and the bank switch program will be loaded from $287 to $2FB of the text buffer.

Appendix I
Command Format Summary

The list below presents the format of each command alongside a statement of its function. Please read the manual for a full explanation of format and function. Many commands have provisions for operands. These operands are indicated by the description inside the parentheses. Refer to the key at the bottom of the page for a further explanation of the operands. If you wish to enter more then one command on a line, please read Section Four, Entering Multiple Commands, in The Bug manual.

EXECUTION COMMANDS

Command	Mnemonic	Description
G (opt add) : (opt cnt)	Go	Standard execution, set P.C.
F (add) : (opt cnt)	Fast	Begin full speed execution
S: (opt cnt)	Step	Begin step by step execution
ESC	Escape	Interrupt execution

SPEED CONTROL COMMANDS

Command	Mnemonic	Description
J: (val)	JSR	Conditional fast execution
Q		Controlled display
I	Instruct	Instructional display
N	Non-Ins	No instructions displayed
B (opt add):opt #	Breakpt	Set a memory loc. breakpoint
T (reg)(symbol)(val)	Test	Set the register test breakpt.
TP (flag/val list)	Tst Pstn	Status reg position, part of T

DISPLAY COMMANDS

Command	Mnemonic	Description
W	When	Display breakpoints
D	Display	Display contents of registers
P (opt add)	Part	Disp. eight memory loc.
Z (opt add)	Zone	Extended memory dump
L (opt add)	List	List disassembly
C	Copy	Print hard copy of displays
CTRL-T	Text	Screen in text mode

CTRL-G	Graphic	Screen in graphic mode
CTRL-F	First	Screen in primary page
CTRL-S	Second	Screen in secondary page
CTRL-N	Normal	Screen in primary page, text
&		Display $3C-$3D
& P		Display $3E-$45

MODIFICATION COMMANDS

Command	Mnemonic	Description
!		Reset stack pointer
R (reg) : (val list)	Register	Modify register value
@ (opt add) : (val list)	At loc	Modify memory at an address
CTRL-W	Write	Write enable for language card
CTRL-P	Protect	Write protect language card
CTRL-K	Check	Checksum against The Bug

EXIT OR RESTART

Command	Mnemonic	Description
E	Exit	Exit to BASIC or DOS
M	Monitor	Access monitor
CTRL-D (instruction)	DOS	Enter a DOS instruction
RESET	Reset	Access coldstart routine
&		Warmstart from Applesoft
CALL 1013		Warmstart from BASIC
CTRL-Y		Warmstart from monitor
CTRL-Q	Quit	Quit The Bug

EXPLANATION OF TABLE ABBREVIATIONS

(add)	A hexadecimal address between $0-$FFFF
(opt add)	An optional hexadecimal address between $0-$FFFF
(val list)	A list of hexadecimal values between $0-$FFFF Each must be separated by a space or a comma
(reg)	A letter A, X, Y, S, or P designating a register
(#)	A number between one and eight
(symbol)	One of the symbols > = < , or a combination
(flag/val list)	One or more of the flags Z, V, C, D, I, B, or N each accompanied by a value of either 1 or 0, and each such pair separated from the next by a comma
(opt cnt)	A hexadecimal value between $1-$FF
(val)	A hexadecimal number between $0-$FF

Appendix II
The Bug Locations

LOC	NAME	DISPL	HIMEM LOC	LCARD LOC
$20	WNDLEFT	$8A1	$90A1	$D8A1
$21	WNDWDTH	$8A2	$90A2	$D8A2
$22	WNDTOP	$8A3	$90A3	$D8A3
$23	WNDBTM	$8A4	$90A4	$D8A4
$24	CH	$8A5	$90A5	$D8A5
$25	CV	$8A6	$90A6	$D8A6
$28	BASL	$8A7	$90A7	$D8A7
$29	BASH	$8A8	$90A8	$D8A8
$2A	BAS2L	$A89	$90A9	$D8A9
$2B	BAS2H	$8AA	$90AA	$D8AA
$2F	LENGTH	--------	---------	----------
$33	PROMPT			
	(BUG'S)	$893	$9093	$D893
	(PROGS)	$894	$9094	$D894
$35	-------	$000	$8800	$D000
$36	CSWL			
	(USER'S)	$001	$8801	$D001
	(MON'S)	$8AB	$90AB	$D8AB
$37	CSWH			
	(USER'S)	$002	$8802	$D002
	(MON'S)	$8AC	$90AC	$D8AC
$38	KSWL			
	(USER'S)	$003	$8803	$D803
	(MON'S)	$8AD	$90AD	$D8AD
$39	KSWH			
	(USER'S)	$004	$8804	$D004
	(MON'S)	$8AE	$90AE	$D8AE
$3A-$3B	PCL-PCH	--------	---------	----------
$3C	A1L	$8D2	$90D2	$D8D2
$3D	A1H	$8D3	$90D3	$D8D3
$3E	A2L	$8D4	$90D4	$D8D4
$3F	A2H	$8D5	$90D5	$D8D5
$40	A3L	$8D6	$90D6	$D8D6
$41	A3H	$8D7	$90D7	$D8D7
$42	A4L	$8D8	$90D8	$D8D8
$43	A4H	$8D9	$90D9	$D8D9
$44	A5L	$8DA	$90DA	$D8DA
$45	A5H	$8DB	$90DB	$D8DB
$4C-$4D	HIMENI	--------	--------	----------
$73-$74	HIMENA	--------	---------	----------

LOC	NAME	DISPL	HIMEM LOC	LCARD LOC
$100	STACK BEG	$D00	$9500	$DD00
$1FF	STACK END	$DFF	$95FF	$DDFF
$200	INPUT BUF			
	(BEG)	$CC1	$94C1	$DCC1
	(END)	$CE9	$94E9	$DCE9
---------	COLDST ROUT			
	(NORM)	$005	$8805	
	(LX)	$2B9	(absolute)	
	(L)	$032	$9532	
---------	WARMST			
	ROUT	$07F	$887F	
	(NORM)	$2C9	(absolute)	
	(LX)	$043	$9543	
	(L)			